REMARKS:

Claims 1-57 were presented for examination and were pending in this application. In an Official Action dated December 3, 2008, claims 1-57 were rejected. Applicants thank Examiner for examination of the claims pending in this application and address Examiner's comments below.

Applicants herein amend claims 1, 17, 19, 46 and 57. These changes are believed not to introduce new matter, and their entry is respectfully requested. The claims have been amended to expedite the prosecution of the application in a manner consistent with the Patent Office Business Goals, 65 Fed. Reg. 54603 (Sept. 8, 2000). In making these amendments, Applicants have not and do not narrow the scope of the protection to which Applicants consider the claimed invention to be entitled and do not concede that the subject matter of such claims was in fact disclosed or taught by the cited prior art. Rather, Applicants reserve the right to pursue such protection at a later point in time and merely seek to pursue protection for the subject matter presented in this submission.

Based on the above Amendment and the following Remarks, Applicants respectfully request that Examiner reconsider all outstanding objections and rejections, and withdraw them.

Objections to the Claims

The Examiner has objected to claim 57 because of informalities. Applicants have amended claim 57 accordingly. Thus, Applicants respectfully submit that all informalities with respect to claim 57 have been resolved.

Response to Rejections Under 35 USC § 103(a)

The Examiner rejects claims 1, 29-33 and 42-47 under 35 USC §103(a) as allegedly being unpatentable over U.S. Patent No. 6,374,286 to Gee et al. ("Gee"). This rejection is respectfully overcome in view of the amended claims.

Claim 1 recites:

> a hardware thread scheduler for identifying which of said program threads said processor executes and configurable to allocate available processing time of the processor among at least the first and second program threads by causing thread-switching between execution of the first program thread directly to execution of the second program thread at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles.

The hardware thread scheduler recited in claim 1 is configurable so the processor switches from execution of one thread directly to execution of another thread according to a predetermined fixed schedule. This beneficially provides a predictable execution time for threads, allowing each thread to be executed for a predetermined number of instruction cycles specified by the predetermined fixed schedule. Additionally, by switching directly from the first thread to the second thread, the hardware thread scheduler reduces the overhead associated with switching between threads, allowing more time for execution of instructions from different threads. *See* spec., page 25, lines 8-13; page 27, line 4 to page 28, line 9.

Gee uses a single Java embedded microprocessor (JEM) to execute multiple Java Virtual Machines (JVMs) by assigning each JVM a fixed area of memory and a fixed amount of time in which to operate. After the fixed amount of time, the JEM invokes a master JVM to handle system duties and subsequently invoke another JVM. Gee, col. 3, lines 50-65.

When switching between JVMs, Gee does not directly transfer control of the JEM from one context to another, but switches from the currently operating JVM to a "master" JVM during a time period called an "interstice." During the interstice, the master JVM performs housekeeping duties and handles system interrupts. Once the interstice completes, the master JVM starts a proxy thread associated with the next JVM to be operated. The proxy thread checks the status of the associated JVM to determine whether or not the associated JVM is ready for operation. Gee, col. 3, line 61-col. 4, line 3. To switch between virtual machines, such as two JVMs, the JEM executes thread control blocks (TCBs) and executive control blocks (ECBs) constructed to look like JAVA objects to simplify manipulation by JAVA software. Gee, col. 19, lines 40-59.

Execution of the ECB causes "interstitial" activity when switching between JVMs, which consumes a number of cycles for system maintenance prior to executing a later JVM, as illustrated in FIG. 14. Gee, col. 23, line 65 to col. 24, line 13; FIG. 14. This "interstitial" activity performs various system-level functions for the JEM, so Gee does not switch "between the first thread directly to the second thread at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles," as claimed. Rather, Gee switches from a first JVM to a system maintenance process and then switches from the system maintenance process to a second JVM for execution. When the system disclosed in Gee switches between contexts, the system maintenance process performs housekeeping operations and services interrupts that had occurred before beginning initialization of the new context. Gee, col. 23,

lines 30-33. While the claimed invention efficiently uses processor resources by directly switching from execution of the first thread to execution of the second thread, Gee performs system maintenance functions prior switching contexts, introducing additional processor overhead when switching between contexts.

Gee discloses that an ECB using a "piano roll" scheduler to process periodic threads with a list of entries which are periodically relayed in a fixed order responsive to receiving an interrupt signal. Gee, col. 20, lines 44-53. While the "piano roll" schedule describes an execution sequence, a master JVM controls execution of different partitions including different JVMs. Gee, col., 23, lines 19-29. As JEM processor resources are allocated among different partitions, the master JVM is invoked when resources are allocated to prepare the JEM processor for execution of a different thread. Gee, col. 23, lines 30-37. Additionally, Gee discloses starting a proxy thread prior to executing the application thread, or JVM, to prepare the JEM for running the application thread, or JVM, which sets event flags and monitors for interrupts applicable to the context being run. Gee, col. 21, lines 56-60. Executing the master JVM between threads causes gaps between execution of JVMs where only system functions not associated with a specific JVM, such as interrupt handling, are performed. Gee, col. 23, lines 30-43; col. 23, line 65 to col. 24, line 13; col. 25, line 24 to col. 26, line 7; FIG. 14.

FIG. 12 of Gee illustrates the disclosed context switching. The partitions (1218, 1224, 1230, 1236, 1242 and 1248) shown in FIG. 12 identify time "slices" where a JVM is executed. When a context switch between JVMs occurs, the operation of one JVM does not immediately follow operation of the preceding JVM. Rather, time gaps (1222, 1228, 1234, 1240, 1246) exist between the partitions allocated for JVM execution. During these time

gaps, only the master JVM operates to service interrupts, schedule the next JVM for execution and perform additional housekeeping activities. Accordingly, when switching between execution of a first JVM and a second JVM, Gee performs two context switches at the end of execution of the first JVM – a first context switch to transfer control from the first JVM to the master JVM and a second context switch to transfer control from the master JVM to the second JVM. Gee, col. 24, lines 4-13; FIG. 12. In contrast, the claimed hardware thread scheduler causes a single thread-switch "between execution of the first program thread directly to execution of the second program thread at a fixed time according to a predetermined fixed schedule," allowing execution of a different thread after a single context switch.

In the Office Action, the "proxy thread" used by Gee to checks the status of the associated JVM to determine whether or not the associated JVM is ready for operation is described as "part of its respective thread." *See* Office Action dated December 3, 2008, page 48. In supporting this statement, FIGS. 14 and 18B of Gee are cited. However, Gee provides that the proxy thread is not part of the execution of a JVM, but is performed prior to execution of the JVM to prepare the JVM for execution. Gee, col. 25, lines 56-65. In FIG. 14, the proxy thread is depicted as distinct from its corresponding JVM and is executed by the master JVM, denoted as JVM0 in FIG. 14, prior to execution of the JVM associated with the proxy thread, shown as JVM2 in FIG. 14. The proxy threads for different JVMs are described as being "in" the master JVM, and the proxy thread execution is illustrated in FIG. 14 as occurring during the time allocated for the master JVM, prior to executing the subsequent JVM. Gee, col. 25, lines 51-55; FIG. 14. Hence, the proxy thread is not part of

the overall JVM, but is executed by the master JVM in preparation for execution of the JVM associated with the proxy thread.

Unlike Gee, the claimed invention directly switches between a first program thread and a second program thread. For example, a context number is passed with an instruction through an execution pipeline. The context number determines context registers from which the program counter is loaded, context registers from which register values are loaded or context registers to which register values are saved. As the context number identifies the collection of storage devices, such as registers, describing the current processor state, switching between different contexts is a matter of using a different context number. By using a different context number, the execution pipeline accesses storage devices associated with the different context number, allowing direct switching from a first context, or thread, to a second context, or thread, without performing system maintenance functions before exeucting the second context, or thread. *See* spec. page 17, line 19 to page 18, line 7. Associating a context number with an instruction allows for contexts to be switched between instructions without requiring system mantenence or processor configuration, unlike Gee.

Accordingly, Gee does not switch "between execution of the first program thread directly to execution of the second program thread." Rather, Gee switches from execution of a first JVM to execution of a system maintenance process then switches from execution of the system maintenance process to a second JVM. The system maintenance process is not a program thread, but merely prepares the JEM processor to subsequently execute a program thread, so there is no direct switching between JVMs, much less direct switching between program threads, disclosed in Gee. In contrast to the disclosure of Gee, the claimed invention fetches the thread identified by a program counter from memory to begin executing

the identified thread in the following cycle, without allocating resources to system maintenance prior to thread execution. *See* specification, page 27, line 21 to page 28, line 9.

Hence, Gee fails to disclose "causing thread-switching between execution of the first program thread directly to execution of the second program thread at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles," as claimed. Rather, Gee switches from execution of a first JVM to execution of system maintenance functions during an interstice and then again switches to execution of a second JVM, providing additional delay in switching between JVMs by performing system maintenance functions prior to executing the later JVM.

Similarly, claim 46 recites:

> switching the processor from the first thread state to the second thread state by coupling the execution pipeline from the first set of data storage devices to the second set of storage devices via the hardware thread selector at a fixed time according to a predetermined fixed execution schedule, said execution schedule specifying that the processor should directly switch to the first thread state from the second thread state every first number of cycles and that the processor should directly switch to the second thread state from the first thread state every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles. (emphasis added)

As discussed above, the system disclosed by Gee switches from execution of a first JVM to execution of a system maintenance process then switches from execution of the system maintenance process to a second JVM. This requires a minimum of two context switches to modify the JVM being executed – a first context switch to transfer control from the first JVM to the system maintenance process and a second context switch to transfer control from the

system maintenance process to the second JVM. Gee, col. 24, lines 4-13; FIG. 12. Hence, all arguments advanced above with respect to claim 1 are also applicable to claim 46, as amended. Thus, Gee fails to disclose that "said execution schedule specifying that the processor should directly switch to the first thread state from the second thread state every first number of cycles and that the processor should directly switch to the second thread state from the first thread state every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles" as recited in claim 46.

Additionally, Gee fails to disclose "coupling the execution pipeline from the first set of data storage devices to the second set of storage devices via the hardware thread selector at a fixed time according to a predetermined fixed execution schedule," as claimed. Gee discloses execution of various software JVMs by a single JEM, so the coupling or an execution pipeline to different data storage devices is not disclosed in Gee, which performs a different software process using the JEM processor. While Gee discloses storing thread control blocks (TCBs) which look similar to a Java object, these TCBs are software constructions, and there is no disclosure that "a first set of data storage devices" and "a second set of data storage devices" are used to maintain a first program thread and a second program thread. FIG. 2 of Gee shows the internal construction of the JEM processor and at most provides a control-store ROM for microinstructions and a ROM code memory for storage of macro level instructions. Gee, FIG. 2; col. 9, lines 5-15. However, these storage devices do not store different program threads. Rather, the macro-level instructions in the ROM code memory point to specific locations in the control-store ROM including the microinstructions for executing the JVM described by the macro-level instructions. Further, the only reference to pipelining in Gee refers to loading the output of the control store into

the microinstruction register at the beginning of each micro cycle to process the content of the microinstruction register while fetching the next microinstruction. Gee, col. 9, lines 32-38. This high-level disclosure of the concept of pipelining does not disclose an "execution pipeline," as recited in claim 46. There is no description of an instruction pipeline or description of stages in an execution pipeline in Gee, much less a description of coupling an execution pipeline to different data storage devices based on a predetermined fixed execution schedule. Accordingly, there is no coupling of "the execution pipeline from the first set of data storage devices to the second set of storage devices via the hardware thread selector at a fixed time according to a predetermined fixed execution schedule," as claimed.

Hence, amended claims 1 and 46 are patentably distinct from the cited reference for at least the reasons described above.

As claims 29-32 and 42-45 are dependent on claim 1, all arguments advanced above with respect to claim 1 are hereby incorporated so as to apply to claims 29-32, 42-45.

As claim 47 is dependent on claim 46, all arguments advanced above with respect to claim 46 are hereby incorporated so as to apply to claim 47.

Applicants respectfully submit that for at least these reasons claims 1, 29-32, 42-47 are patentably distinguishable over the cited reference. Thus, Applicants kindly request withdrawal of these rejections.

The Examiner rejects claims 1, 29-33, 42-43 and 45-47 under 25 USC § 103(a) as allegedly being unpatentable over U.S. Patent No. 6,076,157 to Borkenhagen et al. ("Borkenhagen") in view of Gee. This rejection is respectfully traversed.

Claim 1 recites:

> a hardware thread scheduler for identifying which of said program threads said processor executes and configurable to allocate available processing time of the processor among at least the first and second program threads by causing thread-switching between execution of the first program thread directly to execution of the second program thread at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles. (emphasis added)

The hardware thread scheduler recited in claim 1 is configurable so the processor switches from execution of one thread directly to execution of another thread according to a predetermined fixed schedule. This beneficially provides a predictable execution time for threads, allowing each thread to be executed for a predetermined number of instruction cycles specified by the predetermined fixed schedule. Additionally, by switching directly from the first thread to the second thread, the hardware thread scheduler reduces the overhead associated with switching between threads, allowing more time for execution of instructions from different threads. *See* spec., page 25, lines 8-13; page 27, line 4 to page 28, line 9.

In contrast, Borkenhagen discloses a system and method for data processing in a multithreaded processor where "[t]he thread switch logic has a time-out register which forces a thread switch when execution of the active thread in the multi-threaded processor exceeds a programmable period of time." Borkenhagen, Abstract. The time-out register in Borkenhagen is used "to force a thread switch to the dormant thread after some time if no useful processing is being accomplished to prevent the system from hanging." Borkenhagen, col. 14, lines 49-51. This forced thread switch prevents a thread from "spinning in a loop unable to do useful work" because it is unable to acquire ownership of, or access to, a necessary resource. Borkenhagen, col. 14, lines 31-34. Thus, the time-out register in

Borkenhagen does not specify the processing time allocated to the first and second threads, but rather specifies a maximum time the first and second threads can be inactive before forcing a thread switch.

As the Examiner admits, Borkenhagen does not disclose "a hardware thread scheduler for identifying which of said program threads said processor executes and configurable to allocate available processing time of the processor among at least the first and second program threads by causing thread-switching at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles." However, as discussed above, Gee does not remedy the deficient disclosure of Borkenhagen.

As discussed above, the system disclosed by Gee switches from execution of a first JVM to execution of a system maintenance process then switches from execution of the system maintenance process to a second JVM. This requires a minimum of two context switches to modify the JVM being executed – a first context switch to transfer control from the first JVM to the system maintenance process and a second context switch to transfer control from the system maintenance process to the second JVM. Gee, col. 24, lines 4-13; FIG. 12. As this "interstitial" activity performs various system functions for the JEM, there is an overhead associated with switching between contexts introduced by the "interstitial" activity, requiring additional context switches when modifying the JVM being executed. At most, the combination of Gee and Borkenhagen would allow switching from a first JVM to

performing system functions to a second JVM at a specified time interval, which is unlike the claimed invention.

Similarly, claim 46 recites:

> switching the processor from the first thread state to the second thread state by coupling the execution pipeline from the first set of data storage devices to the second set of storage devices via the hardware thread selector at a fixed time according to a predetermined fixed execution schedule, said execution schedule specifying that the processor should directly switch to the first thread state from the second thread state every first number of cycles and that the processor should directly switch to the second thread state from the first thread state every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles. (emphasis added)

As discussed above, Borkenhagen discloses using a time-out register to force a thread switch when the currently executing thread is inactive or fails to produce a useful result for a defined number of cycles to prevent a background thread from spinning in a loop. Thus, the time-out register in Borkenhagen specifies a maximum number of cycles in which a thread can be inactive or fail to produce useful output before forcing a thread switch. The time-out register in Borkenhagen allows a thread to continue executing indefinitely and does not switch threads until the executing thread fails to produce useful output or becomes inactive for a defined number of cycles. Thus, Borkenhagen fails to disclose that "the processor should directly switch to the first thread state from the second thread state every first number of cycles and that the processor should directly switch to the second thread state from the first thread state every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles" as recited in claim 46.

Gee does not remedy this deficient disclosure, as discussed above, but discloses using software processes executed by the JEM to switch between JVMs. Although Gee uses "piano roll" schedulers to describe thread selection order, context switches between threads

cause "interstitial" activity when switching between JVMs, consuming a number of cycles for performing system housekeeping or resource allocation in preparation for executing a different JVM, as depicted in FIG. 14 of Gee. Gee, col. 23, line 65 to col. 24, line 13; FIG. 14. As this "interstitial" activity performs various system functions for the JEM, it introduces an overhead with switching between and does not directly switch from executing a first thread to executing a second thread, as claimed. At most, the combination of Gee and Borkenhagen would allow switching from a first JVM to performing system functions then switching to a second JVM at a specified time interval, which is unlike the claimed invention.

As claims 29-32 and 42, 43 and 45 are dependent on claim 1, all arguments advanced above with respect to claim 1 are hereby incorporated so as to apply to claims 29-32, 42, 43 and 45.

As claim 47 is dependent on claim 46, all arguments advanced above with respect to claim 46 are hereby incorporated so as to apply to claim 47.

Applicants respectfully submit that for at least these reasons claims 1, 29-32, 42, 43 and 45-47 are patentably distinguishable over the cited reference. Thus, Applicants kindly request withdrawal of these rejections.

The Examiner rejects claims 2-4, 13, 16-17, 19-24 and 56-67 under 35 USC § 103(a) as allegedly being unpatentable over U.S. Patent No. 6,542,991 to Joy et al. ("Joy") in view of "Using Horizontal Prefetching to Circumvent the Jump Problem," by McCrackin et al. ("McCrackin"). This rejection is respectfully overcome.

Claim 17 recites:

> a hardware thread scheduler for identifying which of said program threads said pipelined processor executes and configurable to allocate available processing time of the pipelined processor among at least the first and second program threads according to an execution schedule responsive to a context number associated with an instruction controlling whether the execution pipeline retrieves the instruction from the first set of data storage devices or the second set of data storage devices;
>
> wherein said thread selection hardware in the pipelined processor switches from said first thread state directly to said second thread state between consecutive instruction cycles without incurring a time penalty in response to the hardware thread scheduler identifying which of said program threads said pipelined processor executes. (emphasis added)

Similarly, claim 19 recites:

> switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive execution cycle without incurring a time penalty by coupling the execution pipeline from the first set of data storage devices to the second set of data storage devices via the hardware thread selector responsive to a context number associated with an instruction identifying the first set of data storage devices or the second set of data storage devices. (emphasis added)

Switching threads between consecutive instruction cycles beneficially allows switching between one program context and another without incurring any time penalty. Switching from one thread to another between the end of an execution cycle before the beginning of a next consecutive instruction cycle beneficially allows switching to occur without the loss of any execution cycles. To switch between the first and second threads without incurring any time penalty, the claimed invention modifies which set of storage devices provide instructions to the execution pipeline according to a context number associated with an instruction in the first thread or second thread. The context number allows the pipelined processor to directly retrieve data from, or store data to, the storage device associated with the instruction. *See* spec., page 18, lines 1-15.

In contrast, Joy describes conventional event driven thread scheduling in a multithreaded processor. The conventional multithreading described in Joy uses processor resources efficiently when a thread is stalled. See Joy, col. 6, lines 21-24. That is, the multithreading in Joy is for efficient use of the processor, to reduce "wasted cycle times resulting from stalling and idling." See Joy, col. 2, lines 17-19. The thread switch logic of Joy includes "multithreaded-type functionality in response to an exception condition." Joy, col. 15, lines 8-11. Context switches in Joy "typically are made in response to interrupts, including hardware and software interrupts, both internal and external, of a processor." Joy, col. 14, lines 62-63. While Joy's system includes "fast thread-switching," where timing of the thread switching process is described in detail, there is no mention of "consecutive cycles" or "zero-time" switching, as claimed. Joy, col. 3, lines 28-56.

The fastest thread switching described in Joy requires at least one overhead cycle. Joy, col. 16, lines 1-5. As part of this overhead, Joy discloses that "thread switch logic generates the TID signal with a thread switch delay or overhead of one processor cycle." Thus, Joy discloses a very small delay in switching, but nonetheless a delay associated with responding to the generated TID signal. Even the most ambitious portions of Joy teach the need of an overhead of at least one processor cycle caused by Joy's need to, responsive to the TID, save and restore the processor state associated with the currently executed thread and the thread to be executed. Joy, col. 15, lines 1-7. To perform the context switch responsive to the TID signal, Joy requires that states from the first thread are saved and assigning new states to the second thread. Joy, col. 6, lines 42-48. To save and restore a processor state, a store operation and a load operation are executed to transfer the current state to a memory and load data from a memory to the processor. While Joy provides that the thread switch

logic provides "fast, nanoseconds range, context switching," the context switching in Joy executes instructions to save and restore processor state. Joy, col. 15, lines 55-58.

In contrast, the claimed invention modifies whether the execution pipeline receives instructions from the first or second set of data storage devices. Rather than switch instructions responsive to a generated signal, the claimed invention directly fetches instructions from different storage devices responsive to the execution scheduler. For example, page 26, line 15 through page 27, line 2 of the specification and Figure 8 describe one implementation of the claimed invention. As described in the specification, multiple storage devices, shown in Figure 8 as "Flash A Fetch," "Flash B Fetch," "Flash C Fetch," "Flash D Fetch" and "Shadow SRAM," provide instructions to an execution pipeline based on the output from a hardware thread scheduler, illustrated in Figure 8 as "Post Fetch Select." Hence, the execution pipeline is capable of executing instructions from different threads during each instruction cycle by accessing a different storage device in each instruction cycle. For example, a context number in an instruction identifies the storage device which provides instructions to the execution pipeline, allowing the execution pipeline to directly retrieve the instructions from the identified data storage device. An example of this zero-time context switching is described in the specification at page 18, line 1 to page 19, line 5. Hence, the claimed invention allows retrieval and execution of instructions from different threads by modifying the storage device from which an execution pipeline retrieves instructions, allowing each stage in an execution pipeline to execute instructions from different threads, while Joy requires that threads be loaded into the processor for execution, introducing overhead into thread switching caused by the time necessary for loading and

storing processor state information. Hence, the claimed invention uses the content of the instruction to determine from which data storage device to retrieve instructions.

Joy discloses using thread switch logic to generate a thread switch signal allowing the processor to change the currently executing thread. Joy, col. 16, lines 9-14. The thread switch logic generates a thread identifier associated with a thread, allowing access to a thread using the thread identifier. However, the claimed invention associates a context number with instructions in the first and second threads, allowing modification of the data storage device coupled to an execution pipeline depending on an instruction within a thread. This context number allows the execution pipeline, or stages within the execution pipeline, to directly retrieve data from, or store data to, a particular storage device. This instruction-level association with a storage device is not disclosed in Joy, which at most allows identification of a thread in its entirety using a thread identifier.

Hence, Joy does not switch threads between consecutive instruction cycles "responsive to a context number associated with an instruction controlling whether the execution pipeline retrieves the instruction from the first set of data storage devices or the second set of data storage devices," but performs a context switching operation where processor states are saved and restored to accelerate conventional thread switching to provide context switching in "nanosecond ranges." Joy, col. 7, lines 50-52. By "responsive to a context number associated with an instruction controlling whether the execution pipeline retrieves the instruction from the first set of data storage devices or the second set of data storage devices," the claimed invention allows execution of different instructions from different threads by specifying what set of data storage devices include the instruction associated with the context number. This also allows different stages in the execution

pipeline to operate in separate contexts based on the context number of the instruction residing in the different stages.

Thus, the claimed "hardware thread scheduler for identifying which of said program threads said pipelined processor executes and configurable to allocate available processing time of the pipelined processor among at least the first and second program threads according to an execution schedule responsive to a context number associated with an instruction controlling whether the execution pipeline retrieves the instruction from the first set of data storage devices or the second set of data storage devices" and "thread selection hardware in the pipelined processor switches from said first thread state directly to said second thread state between consecutive instruction cycles without incurring a time penalty in response to the hardware thread scheduler identifying which of said program threads said pipelined processor executes" is not disclosed by Joy.

McCrackin fails to remedy the deficient disclosure of Joy. Rather, McCrackin discloses horizontally prefetching instructions across different instruction streams to rapidly switch between contexts. McCrackin, pg. 1287, Abstract. In McCrackin, streams compete for a bus interface and instruction unit for prefetching and an execution unit for execution by a horizontal demand prefetching (HDP) processor. As more streams are added to the HDP processor, prefetch depth increases, improving processor performance. McCrackin, pg. 1288, § II, col. 1. A stream control unit selects different streams for execution by the HDP processor accounting for priority levels of each stream. McCrackin, pg. 1289, § III, cols. 1-2.

However, McCrackin also fails to disclose a "hardware thread scheduler for identifying which of said program threads said pipelined processor executes and configurable to allocate available processing time of the pipelined processor among at least the first and

second program threads according to an execution schedule responsive to a context number associated with an instruction controlling whether the execution pipeline retrieves the instruction from the first set of data storage devices or the second set of data storage devices," as claimed. McCrackin does not associate a context number with an instruction within a stream to identify the storage device associated with the instruction, but selects between complete streams for execution. The stream control unit distributes processing time among various streams and verifies execution of streams with full prefetch registers. McCrackin, pg. 1288, § II, col. 1. McCrackin makes no disclosure of the instructions within a stream, much less associating a context number with an instruction within the stream to identify a set of storage devices for coupling to the execution pipeline. Coupling of storage devices to an execution pipeline responsive to a context number associated with an instruction in a thread is not disclosed or suggested McCrackin. Therefore, McCrackin also fails to disclose "hardware thread scheduler for identifying which of said program threads said pipelined processor executes and configurable to allocate available processing time of the pipelined processor among at least the first and second program threads according to an execution schedule responsive to a context number associated with an instruction controlling whether the execution pipeline retrieves the instruction from the first set of data storage devices or the second set of data storage devices," as claimed.

While the claimed invention associates a context number with instructions from a thread, the combination of McCrackin and Joy at most allows for switching between threads based on a thread selection signal generated by a thread scheduler or stream control unit. This switching is not based on a storage device associated with instructions within the thread, but is based on a single identifying a thread as a whole, rather than a context number

associated with individual instructions within a thread. The generation of a thread selection signal to select an entire thread is distinct from associating a context number with an instruction within a thread. While the claimed invention allows each stage in an execution pipeline to operate in a different context based on the context number, a thread-level selection signal requires all stages in an execution pipeline to operate in the same context.

Hence, claims 17 and 19 are patentably distinguishable from the cited references, both alone and combination, so reconsideration and withdrawal of their rejection is respectfully requested.

As claims 2-4, 13 and 16 are dependent from claim 17, all arguments advanced above with respect to claim 17 are hereby incorporated so as to apply to claims 2-4, 13 and 16.

As claims 20-24 are dependent from claim 19, all arguments advanced above with respect to claim 19 are hereby incorporated so as to apply to claims 20-24.

Accordingly, for at least the reasons set forth above, claims 2-4, 13, 16-17 and 19-24 are patentable over the cited references, both alone and in combination.

The Examiner rejects claims 5-12, 18 and 25-28 under 25 USC § 103(a) as allegedly being unpatentable over Joy and McCrackin in view of U.S. Patent No. 6,085,215 to Ramakrishnan et al. ("Ramakrishnan"). This rejection is respectfully overcome in view of the amended claims.

As claims 5-12 and 18 are dependent on claim 17, all arguments advanced above with respect to claim 17 are hereby incorporated so as to apply to claims 5-12 and 18.

Ramakrishnan is cited to make up for Joy's lack of "thread identifier for identifying at least one hard-real-time (HRT) thread and at least one non-real-time thread" limitation. Ramakrishnan simply describes a software scheduler that uses a round robin approach to

thread scheduling using conventional context switching. See Ramakrishnan, col. 9, lines 9-10. The switching in Ramakrishnan, like in Joy, is conventional context switching involving "an associated overhead in invoking the new thread." (Ramakrishnan, col. 12, lines 61-62, see also, col. 13, lines 6-7 "avoid time consuming context switching"). Ramakrishnan does not remedy the deficiencies of Joy and McCrackin discussed above. Hence, the combination of Joy and Ramakrishnan still fails to disclose the "hardware thread scheduler for identifying which of said program threads said pipelined processor executes and configurable to allocate available processing time of the pipelined processor among at least the first and second program threads according to an execution schedule responsive to a context number associated with an instruction controlling whether the execution pipeline retrieves the instruction from the first set of data storage devices or the second set of data storage devices" recited in claim 17.

Accordingly, for at least the reasons set forth above, claims 5-12, 18 and 25-28 are patentable over the cited references, both alone and in combination. Thus, Applicants kindly request withdrawal of these rejections.

The Examiner rejects claims 34-41 and 48-55 as allegedly being unpatentable over Gee in view of Ramakrishnan. This rejection is respectfully traversed.

As claims 34-41 are dependent on claim 1, all arguments advanced above with respect to claim 1 are hereby incorporated so as to apply to claims 34-41. As claims 48-55 are dependent on claim 46, all arguments advanced above with respect to claim 46 are hereby incorporated so as to apply to claims 48-55.

Ramakrishnan is cited to make up for Gee's lack of "a thread identifier for identifying at least one hard-real-time (HRT) thread and at least one non-real-time thread" limitation.

Ramakrishnan simply describes a software scheduler that uses a round robin approach to thread scheduling using conventional context switching. See Ramakrishnan, col. 9, lines 9-10. The scheduler in Ramakrishnan also fails to disclose "causing thread-switching between execution of the first program thread directly to execution of the second program thread at a fixed time according to a predetermined fixed schedule." Hence, Ramakrishnan does not remedy the deficiencies of Gee.

Accordingly, for at least the reasons set forth above, claims 34-41 and 48-55 are patentable over the cited references, both alone and in combination. Thus, Applicants kindly request withdrawal of these rejections.

The Examiner rejects claim 14 as allegedly being unpatentable over Joy in view of McCrackin and Borkenhagen. This rejection is respectfully traversed.

As claim 14 is dependent on claim 17, all arguments advanced above with respect to claim 17 are hereby incorporated so as to apply to claim 14.

Borkenhagen is cited to make up for the deficiency of "storing said second thread state of said processor during execution of said first program thread" in Joy. However, the combination of Joy and Borkenhagen still fails to teach or suggest the "between consecutive instruction cycles" switching recited in claim 17. The system disclosed in Borkenhagen also incurs the conventional "latency and performance penalties associated with switching threads." Borkenhagen, col. 15, lines 37-38. Borkenhagen explicitly discloses:

> In the multithreaded processor in the preferred embodiment described herein, this latency includes the time required to complete execution of the current thread to a point where it can be interrupted and correctly restarted when it is next invoked, the time required to switch the thread-specific hardware facilities from the current thread's state to the new thread's state, and the time required to restart the new thread and begin its execution.

Borkenhagen col. 15, lines 38-46. Thus, Borkenhagen also fails to disclose "hardware thread scheduler for identifying which of said program threads said pipelined processor executes and configurable to allocate available processing time of the pipelined processor among at least the first and second program threads according to an execution schedule responsive to a context number associated with an instruction controlling whether the execution pipeline retrieves the instruction from the first set of data storage devices or the second set of data storage devices," so Borkenhagen fails to cure the deficiencies of Joy and McCrackin.

Accordingly, for at least the reasons set forth above, claim 14 is patentable over the cited references, both alone and in combination. Thus, Applicants kindly request withdrawal of this rejection.

The Examiner rejects claim 15 as allegedly being unpatentable over Joy in view of McCrackin in further view of U.S. Patent No. 6,314,511 to Levy et al. ("Levy"). This rejection is respectfully overcome.

As claim 15 is dependent on claim 17, all arguments advanced above with respect to claim 17 are hereby incorporated so as to apply to claim 44.

Levy is cited to make up for Joy's lack of "said first set of data storage devices comprises registers shared by a plurality of threads." However, Levy discloses a method "for freeing a renaming register, the renaming register being allocated to an architectural register by a processor for the out-of-order execution of at least one of a plurality of instructions." Levy, col. 3, lines 27-30. Levy makes no mention of associating a context number with an instruction to modify the storage device coupled to the execution pipeline, but merely discloses a configuration for a "processor with dynamic out-of-order instruction processing

capability." Levy, col. 7, lines 18-19. As Levy also does not disclose "hardware thread scheduler for identifying which of said program threads said pipelined processor executes and configurable to allocate available processing time of the pipelined processor among at least the first and second program threads according to an execution schedule responsive to a context number associated with an instruction controlling whether the execution pipeline retrieves the instruction from the first set of data storage devices or the second set of data storage devices" Levy fails to remedy the deficient disclosure of Joy and McCrackin.

Accordingly, for at least the reasons set forth above, claim 15 is patentable over the cited references, both alone and in combination. Thus, Applicants kindly request withdrawal of this rejection.

The Examiner rejects claims 34-41 and 48-55 as allegedly being unpatentable over Borkenhagen in view of Gee in further view of Ramakrishnan. This rejection is respectfully overcome.

As claims 34-41 are dependent on claim 1, all arguments advanced above with respect to claim 1 are hereby incorporated so as to apply to claims 34-41. As claims 48-55 are dependent on claim 46, all arguments advanced above with respect to claim 46 are hereby incorporated so as to apply to claims 48-55.

Ramakrishnan is cited to make up for Borkenhagen and Gee's lack of "a thread identifier for identifying at least one hard-real-time (HRT) thread and at least one non-real-time thread" limitation. Ramakrishnan simply describes a software scheduler that uses a round robin approach to thread scheduling using conventional context switching. See Ramakrishnan, col. 9, lines 9-10. The scheduler in Ramakrishnan also fails to disclose "causing thread-switching between execution of the first program thread directly to execution

of the second program thread at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles" for allocating thread processing time. Hence, Ramakrishnan does not remedy the deficiencies of the combination of Borkenhagen and Gee.

Accordingly, for at least the reasons set forth above, claims 34-41 and 48-55 are patentable over the cited references, both alone and in combination. Thus, Applicants kindly request withdrawal of these rejections.

The Examiner rejects claim 44 as allegedly being unpatentable over Borkenhagen in view of Gee and in further view of Levy. This rejection is respectfully overcome.

As claim 44 is dependent on claim 1, all arguments advanced above with respect to claim 1 are hereby incorporated so as to apply to claim 44.

Levy is cited to make up for the combination of Gee and Borkenhagen lack of "said first set of data storage devices comprises registers shared by a plurality of threads." However, Levy discloses a method "for freeing a renaming register, the renaming register being allocated to an architectural register by a processor for the out-of-order execution of at least one of a plurality of instructions." Levy, col. 3, lines 27-30. Levy makes no mention of a "hardware thread scheduler for identifying which of said program threads said processor executes and configurable to allocate available processing time of the processor among at least the first and second program threads by causing thread-switching between execution of the first program thread directly to execution of the second program thread at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread

should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles," but merely discloses a configuration for a "processor with dynamic out-of-order instruction processing capability." Levy, col. 7, lines 18-19. As Levy does not disclose "causing thread-switching at a fixed time according to a predetermined fixed schedule," it fails to remedy the deficient disclosure of the combination of Borkenhagen and Gee.

Accordingly, for at least the reasons set forth above, claim 44 is patentable over the cited references, both alone and in combination. Thus, Applicants kindly request withdrawal of this rejection.

New claims 56 and 57 have been added. Support for claims 56 and 57 are found throughout the specification, for example at page 18, lines 1-7. New claim 56 depends from claim 19 as well as recites additional patentable features, such as "communicating the context number associated with the second program thread to the execution pipeline," and "loading instructions from the second set of data storage devices into the execution pipeline." New claim 57 depends from claim 17 and also recites additional patentable features, such as "communicating the context number associated with the second program thread to the execution pipeline," and "loading instructions from the second set of data storage devices into the execution pipeline."

Conclusion

In sum, Applicants respectfully submit that claims 1-57, as presented herein, are patentably distinguishable over the cited references. Therefore, Applicants request reconsideration of the basis for the rejections to these claims and request allowance of them.

In addition, Applicants respectfully invite Examiner to contact Applicants' representative at the number provided below if Examiner believes it will help expedite furtherance of this application.

Respectfully Submitted,
NICHOLAS J. KELSEY, ET AL.

Date: June 1, 2009

By: /Brian G. Brannon/
Brian G. Brannon, Registration No. 57,219
FENWICK & WEST LLP
801 California Street
Mountain View, CA 94041
Phone: (650) 335-7610
Fax: (650) 938-5200
E-Mail: bbrannon@fenwick.com

20880/05093/DOCS/2052938.1